

USAISEC

*US Army Information Systems Engineering Command
Fort Huachuca, AZ 85613-5300*

AD-A268 379



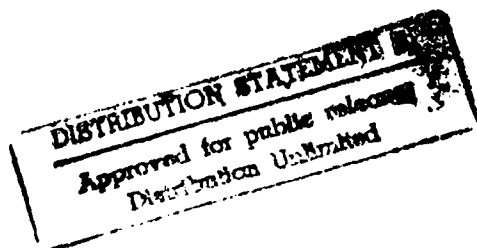
U.S. ARMY INSTITUTE FOR RESEARCH
IN MANAGEMENT INFORMATION,
COMMUNICATIONS, AND COMPUTER SCIENCES

SAMeDL: Technical Report Appendix F – User's Guide Part 2 – ORACLE

ASQB-GI-92-020

September 1992

DTIC
ELECTE
AUG 12 1993
S B D



**AIRMICS
115 O'Keefe Building
Georgia Institute of Technology
Atlanta, GA 30332-0800**

93-18872



3718



REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188
Exp. Date: Jun 30, 1986

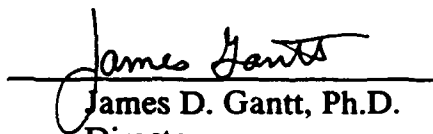
1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED			1b. RESTRICTIVE MARKINGS NONE		
2a. SECURITY CLASSIFICATION AUTHORITY N/A			3. DISTRIBUTION/AVAILABILITY OF REPORT N/A		
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A					
4. PERFORMING ORGANIZATION REPORT NUMBER(S)			5. MONITORING ORGANIZATION REPORT NUMBER(S) N/A		
6a. NAME OF PERFORMING ORGANIZATION	6b. OFFICE SYMBOL (If applicable)		7a. NAME OF MONITORING ORGANIZATION N/A		
6c. ADDRESS (City, State, and Zip Code)			7b. ADDRESS (City, State, and ZIP Code) N/A		
8b. NAME OF FUNDING/SPONSORING ORGANIZATION Software Technology Branch, ARL	8b. OFFICE SYMBOL (If applicable) AMSRL-CI-CD		9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER		
8c. ADDRESS (City, State, and ZIP Code) 115 O'Keefe Bldg. Georgia Institute of Technology Atlanta, GA 30332-0800			10. SOURCE OF FUNDING NUMBERS		
			PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO.
			WORK UNIT ACCESSION NO.		
11. TITLE (Include Security Classification) SAMEDL: Technical Report Appendix F - User's Guide Part 2 - Oracle					
12. PERSONAL AUTHOR(S) MS. Deb Waterman					
13a. TYPE OF REPORT Technical Paper	13b. TIME COVERED FROM Apr 91 TO Sept 92	14. DATE OF REPORT (Year, Month, Day) Sept 15, 1992		15. PAGE COUNT 50	
16. SUPPLEMENTARY NOTATION					
17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)		
FIELD	GROUP	SUBGROUP	Ada Database Access, SAMEDL, Ada extension module, SQL		
19. ABSTRACT (Continue on reverse if necessary and identify by block number)					
<p>This report details the research efforts into the SQL Ada Module Database Description Language (SAMEDL). Four compilers are presented (Oracle, Informix, XDB, and Sybase) that allow Ada application programs to access database using a standard SQL query language. Copies of the compiler can be obtained from the DoD Ada Joint Program Office 703/614-0209.</p>					
20. DISTRIBUTION/AVAILABILITY OF ABSTRACT <input checked="" type="checkbox"/> UNCLASSIFIED/UNLIMITED <input type="checkbox"/> SAME AS RPT. <input type="checkbox"/> DTIC USERS			21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED		
22a. NAME OF RESPONSIBLE INDIVIDUAL LTC David S. Stevens			22b. TELEPHONE (Include Area Code) (404) 894-3110		22c. OFFICE SYMBOL AMSRL-CI-CD

This research was performed by Statistica Inc., contract number DAKF11-91-C-0035, for the Army Institute for Research in Management Information, Communications, and Computer Sciences (AIRMICS), the RDTE organization of the U. S. Army Information Systems Engineering Command (USAISEC). This final report discusses a set of SAMeDL compilers and work environment that were developed during the contract. Request for copies of the compiler can be obtained from the DoD Ada Joint Program Office, 703/614/0209. This research report is not to construed as an official Army or DoD Position, unless so designated by other authorized documents. Material included herein is approved for public release, distribution unlimited. Not protected by copyright laws.

THIS REPORT HAS BEEN REVIEWED AND IS APPROVED



Glenn E. Racine, Chief
Computer and Information
Systems Division



James D. Gantt, Ph.D.
Director
AIRMICS

DTIC QUALITY INSPECTED 3

Accession For	
NTIS GRA&I	<input checked="" type="checkbox"/>
DTIC TAB	<input type="checkbox"/>
Unannounced	<input type="checkbox"/>
Justification	
By	
Distribution/	
Availability Codes	
Dist	Avail and/or Special
A-1	

SAMeDL Development Environment User Manual

(Oracle/386PC/Interactive UNIX/Alsys)

Intermetrics, Inc.

Document	IR-VA-028-1
Date	01-September-1992

Published by
Intermetrics, Inc.
733 Concord Avenue, Cambridge, Massachusetts 02138

Copyright (c) 1992 by Intermetrics, Inc.

This material may be reproduced by or for the U.S. Government pursuant to the copyright license under the clause at DFARS 252.227-7013 (Oct. 1988).

Table Of Contents

Chapter 1	About This Manual.....	1
	1.1 Purpose	1
	1.2 Organization	1
	1.3 Syntax Conventions.....	2
	1.4 References	2
Chapter 2	SDE Overview	5
	2.1 The SDE SAMeDL Compiler.....	5
	2.2 The SDE Module Manager.....	6
Chapter 3	SDE Library File System	7
	3.1 Overview Of SDE Libraries	7
	3.2 Core Library Files.....	8
	3.3 Modules and Interface Files	10
	3.4 Miscellaneous Temporary Files.....	10
	3.5 Standard SAMeDL Modules and Ada Support Packages	11
Chapter 4	Getting Started With SDE	13
	4.1 Creating A Database.....	13
	4.2 Creating An SDE Library	13
	4.3 Compiling A SAMeDL Source File	14
	4.4 Creating An Ada Application Program	16
Chapter 5	Building Ada/SQL Interfaces With SAMeDL	23
	5.1 Overview Of The SAMeDL Compiler	23
	5.2 SAMeDL Compiler Invocation	23
	5.3 Using the Compiler-Generated Interface.....	24
	5.4 Compiler Directives.....	26
	5.4.1 Reference Directive	26
	5.4.2 Owner Directive	27
Chapter 6	Implementation Dependent Features.....	29
	6.1 SAMeDL Language Limitations Under Oracle.....	29
	6.2 SAMeDL Extensions For Oracle.....	29
	6.3 Troubleshooting Common System Errors	30
Chapter 7	Tool Limitations.....	31
	7.1 SAMeDL Compiler Limitations.....	31
	7.2 SDE Module Manager Limitations.....	31
Chapter 8	SDE Command Reference Manual Pages	33
	8.1 samedl.....	34
	8.2 sde.cleanlib	36
	8.3 sde.creatar	37
	8.4 sde.creatlib.....	38
	8.5 sde.ls	39
	8.6 sde.mkscript.....	41
	8.7 sde.purge.....	43
	8.8 sde.rm	44
	8.9 sde.rmlib	45

Chapter 1 About This Manual

1.1 Purpose

The purpose of this manual is to describe the features of the Intermetrics' SAMeDL Development Environment (SDE) for the Oracle Database Management System on the 386PC platform with Interactive UNIX and Alsys Ada. The language supported is defined in the *SAMeDL Language Reference Manual* [LRM]. This user's manual is not intended to be a language tutorial for SAMeDL. In addition, it is assumed that you have an underlying working knowledge of Oracle [Oracle] and the Ada standard [Ada].

1.2 Organization

The organization of this document is as follows:

- Chapter 2, *SDE Overview*, briefly describes the SDE components, what each component is used for, and how the components relate to each other.
- Chapter 3, *SDE Library File System*, contains an overview of libraries and how SDE uses them.
- Chapter 4, *Getting Started With SDE*, demonstrates a simple scenario, providing enough information for users to get started developing Ada/SQL interfaces with SAMeDL.
- Chapter 5, *Building Ada/SQL Interfaces With SAMeDL*, provides detailed information on how to generate Ada/SQL interfaces using the SAMeDL compiler, and also outlines the procedures that should be followed for including generated interfaces in an Ada application program.
- Chapter 6, *Implementation Dependent Features*, discusses SAMeDL features which are dependent on the Oracle DBMS implementation.
- Chapter 7, *Tool Limitations*, outlines general restrictions and tool limitations imposed by the current release of the SAMeDL Development Environment for Oracle/386PC/Interactive UNIX/Alsys.
- Chapter 8, *SDE Command Reference Manual Pages*, contains a detailed reference for each command in SDE.

1.3 Syntax Conventions

The following explains the notational conventions used in SDE command syntax throughout this document:

xyz Items expressed in lower-case italic letters are used to represent user-supplied names. You should substitute an appropriate value. For example,

pathname

would mean that you should specify the text that represents a file or directory pathname.

[] Brackets are used to denote items that are optional. For example,

`sde.cleanlib [pathname]`

means that you may specify the command with or without supplying a *pathname*.

... An ellipsis indicates that you may optionally repeat the preceding item one or more times. For example,

`module_name ...`

means that a series of module names can follow the one listed.

Unless otherwise noted, you may specify options on a SDE command in any order. Also, option keywords are not case sensitive and may be truncated as long as the resulting abbreviation is unambiguous. For example, the following two commands are equivalent:

```
sde.ls -l my_library -v my_def_module
sde.ls -Verbose -Library my_library my_def_module
```

1.4 References

[Ada] *Reference Manual for the Ada Programming Language*, Ada Joint Program Office, 1983.

[AdaRef] *FirstAda Ada Software Engineering Environment: Application Developer's Guide and Appendix F version 4.4*, Alsys, Inc, 1990.

[LRM] *SAMeDL Language Reference Manual*, Intermetrics, Inc., IR-VA-011-1, 07 July 1992.

[Oracle] *ORACLE RDBMS Database Administrator's Guide*, Oracle Corporation, October 1990. Part No. 3601-V6.0.

[OraESQL] Programmer's Guide to the *ORACLE Precompilers*, Oracle Corporation, September 1991. Part No. 5315-V1.3.

[SAMEGuide] *Guidelines for the Use of the SAME*, Marc Graham: Software Engineering Institute/Carnegie Mellon University, Technical Report CMU/SEI-89-TR-16, May 1989.

[SDEInst] *SAMeDL Development Environment Installation Guide*, Intermetrics, Inc., IR-VA-026-2, 01 September 1992.

Chapter 2 SDE Overview

The SAMeDL Development Environment (SDE) provides you with a software environment for developing Ada/SQL interfaces through the use of SAMeDL. The SDE toolset consists of a compiler, which processes SAMeDL source files to generate Ada/SQL interfaces, and the Module Manager, which assists you with SDE library management and other facets of interface development.

The SDE toolset includes the following:

samedl	invoke the SAMeDL compiler
sde.cleanlib	reinitialize an SDE library
sde.creatar	create a library archive file for compiled concrete modules
sde.creatlib	create an SDE library
sde.ls	list compiled SAMeDL modules
sde.mkscript	generate an Ada compilation script file for an interface file
sde.purge	remove out of date files from an SDE library
sde.rm	remove a SAMeDL module from an SDE library
sde.rmlib	remove an SDE library

2.1 The SDE SAMeDL Compiler

The SAMeDL compiler processes SAMeDL source files and generates interface files representing the prescribed Ada/SQL interface.

Like an Ada compiler which deals with compilation units, the SAMeDL compiler works with *modules*, which are the smallest pieces of code that can be successfully compiled and shared. A SAMeDL source file may consist of one or more modules.

In SAMeDL, there are three types of modules. A module may be either a definitional module containing shared definitions, a schema module containing table, view, and privilege definitions, or an abstract module containing local definitions and procedure/cursor declarations.

The SAMeDL compiler will generate interface files for each definition module (in the form of an Ada package specification/body pair) and each abstract module (in the form of a layered interface consisting of an Ada package specification/body pair and an object code file generated from a C with embedded SQL file). No interface files are generated for schema modules. The generated interface files collectively represent the Ada/SQL interface you would use in your Ada application program.

SAMeDL is analogous to Ada in that it also has the concept of *separate compilation*. SAMeDL modules may use (through the use of *context clauses*) information contained in other modules that you have previously compiled. All separate compilation information is kept in ordinary host file system directories and files. These files/directories along with any generated interface files are organized into an *SDE library*, which again is somewhat similar to the development library concept used by most Ada development systems.

As in the case of most language compilers, the SAMeDL compiler will perform the appropriate syntactic and semantic error checking. All error messages are reported to the standard output device. You may also optionally specify that a source listing file be generated in which case, if you had any errors, the errors would be interleaved with the SAMeDL source code in your listing.

2.2 The SDE Module Manager

The SDE Module Manager is a set of tools which you may use to assist with SDE library management and other facets of interface development. These tools include **sde.cleanlib**, **sde.creatar**, **sde.creatlib**, **sde.ls**, **sde.mkscript**, **sde.purge**, **sde.rm**, and **sde.rmlib**.

sde.cleanlib

sde.cleanlib will allow you to empty an existing SDE library of all compilation information. The command will re-initialize the **names.dbe** and **samedl.dat** files and remove the remaining contents of the **samedl.lib** subdirectory.

sde.creatar

sde.creatar is useful for creating and updating a library archive file that contains the object code files generated from the C with embedded SQL for abstract modules. The object code files can then more easily be included as part of the Ada link step for the application program (as opposed to specifying each of the object code files individually in the link step).

sde.creatlib

sde.creatlib is used to create and initialize a new SDE library. It creates a directory named **samedl.lib** in the library directory, and creates the files **samedl.dat** and **names.dbe** in the **samedl.lib** directory.

sde.ls

sde.ls provides you with a list of the SAMeDL modules compiled in an SDE library. Useful is the **interface** option which will provide information concerning the interface files generated for a module.

sde.mkscript

sde.mkscript will create a template for performing the Ada compilation of the generated Ada interface files (and the units they depend on) for the definitional or abstract modules.

sde.purge

sde.purge will remove all out of date/unused files from an SDE library. These files include temporary files (e.g., those used during compilation) or interface files that have been put out of date due to recompilation of the associated SAMeDL modules. In addition, **sde.purge** will also remove the library state information backup file **samedl.dat.back**.

sde.rm

sde.rm allows you to remove all information and related interface files associated with modules compiled in the SDE library.

sde.rmlib

You use **sde.rmlib** to remove an SDE library and all of the information it contains.

Chapter 3 SDE Library File System

This chapter contains an overview of SDE libraries and the files that comprise them.

3.1 Overview Of SDE Libraries

An SDE library is a host file system directory which acts as a central database of SAMeDL compilation information and related generated interfaces.

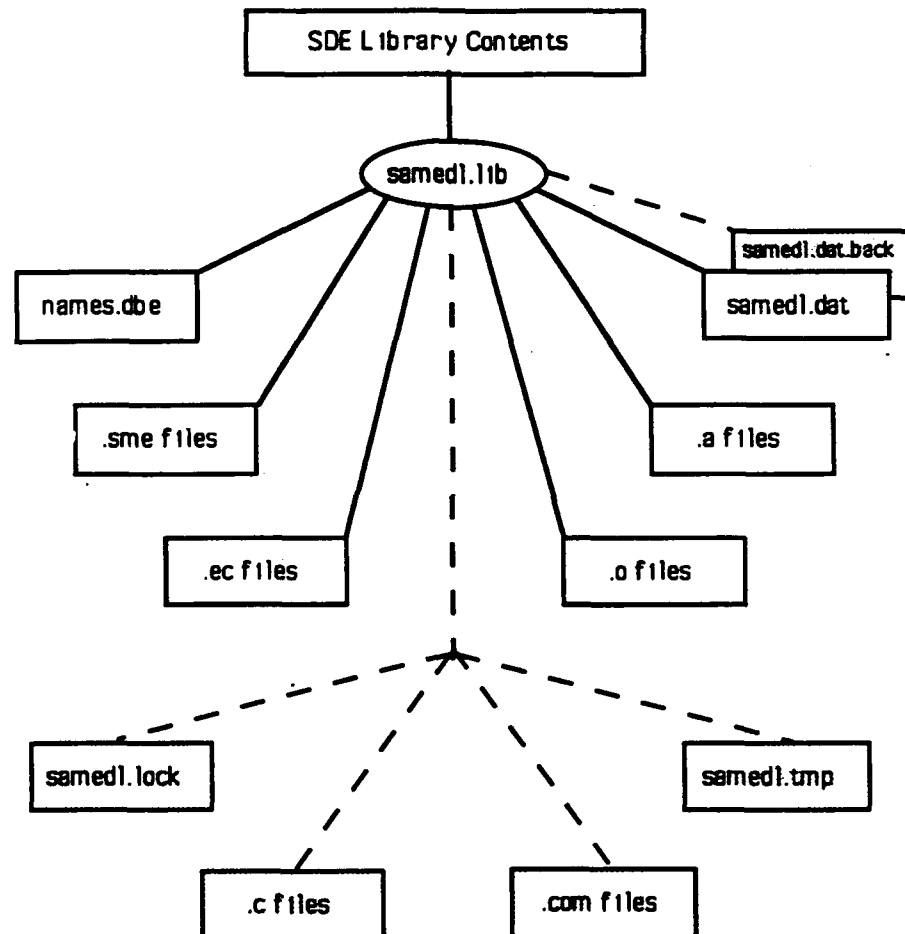


Figure 3.1: Contents of an SDE Library

Every directory representing an SDE library will contain the directory **samedl.lib**. **samedl.lib** in turn contains the files **names.dbc**, **samedl.dat**, and **samedl.dat.back**, and various files ending with **.sme**, **.ec**, **.o**, and **.a** extensions. In addition, there are a variety of temporary files that may appear under **samedl.lib**: **samedl.lock**, **samedl.tmp**, and files ending with **.c** and **.com** extensions.

Note: in general, it is not advisable for you to modify or place any files in the directory **samedl.lib** that are not otherwise generated by SDE. In particular, **sde.rmllib** and **sde.cleanlib** do the equivalent of a UNIX **"rm -r samedl.lib"** as part of their operation.

3.2 Core Library Files

When you initially create a new SDE library (via `sde.creatlib`) or "clean" an existing SDE library (via `sde.cleanlib`), the directory `samedl.lib` will only contain the following core library files: `samedl.dat` and `names.dbc`. In certain circumstances, a backup file for `samedl.dat` named `samedl.dat.back` will also be present.

samedl.dat

`samedl.dat` is the net disk data file for the library. It contains a series of records, each record containing the data for a single node in the internal representation of the dependency tree. The information in the file is in text format, that can be read/written by the SDE module manager and the SAMeDL compiler.

The internal representation of the dependency information is tree-like. Each node in the tree represents a file in the SAMeDL system, and has information about all nodes that are dependent on it and nodes that it depends on (called *CaredAboutBy* and *CaresAbout* arcs respectively). Each node also contains the time it was created, the external source file it was created from, the name of the source file saved in the library and the name of the library file that the generated code resides in.

Nodes are given node numbers that uniquely identify them. This practice facilitates saving the tree to the designated disk file and reading it back because pointers do not need to be included in the disk file. It also facilitates the use of uniform data structures for the internal representation because variable length records do not need to be used. Instead, lists are maintained off each node that contain the node numbers of the nodes that the node depends upon, or is depended upon by.

The records in the disk data have the following fields:

Node Number	number of the node that specifies the unit
Node Type	the type of file this node points to
Unit Name	name of the compiled unit
Time Entered	time the unit was entered into the library
Library File	name of file saved in library
External File	pathname of file that the node was generated from
Cares About Arc Num	number of cares about arcs from this node
Cares About Arc List	list of cares about arcs from this node
Cared About By Arc Num	number of care about arcs to this node
Cared About By Arc List	list of care about arcs to this node

The records in the disk data file are written out in text form, one after the other with a special character separating each node.

samedl.dat.back

The **samedl.dat.back** file is a backup copy of the **samedl.dat** file that the SAMeDL compiler and the **sde.rm** command make before they change the **samedl.dat** file. **samedl.dat.back** will contain the prior library state information and thus will allow you to undo the effects of the last **samedl** or **sde.rm** command (provided that a **sde.purge** command has not been since executed; see below). In order to restore the library back to its prior state, you should go to the **samedl.lib** directory corresponding to your SDE library, remove the existing **samedl.dat** file, and rename (using the UNIX **mv** command for example) **samedl.dat.back** to **samedl.dat**. Note that because of the semantics of the **sde.purge** command, an SDE library may not be restored if the library has been purged.

names.dbe

The **names.dbe** file is a text file that maintains two integer counters used internally by the compiler to keep track of procedures and variables across separate compilations.

Chapter 4 Getting Started With SDE

This chapter presents some basic scenarios for using the SAMeDL Development Environment: creating an SDE library, compiling a SAMeDL source file, and creating an Ada application program which uses the SAMeDL compiler generated modules to interface with the database environment. The scenarios have intentionally been kept simple; details are deferred to later sections of this manual.

Suppose you want to design an Ada application program which interacts with a database environment. The basic steps are:

1. Create the Oracle Database that the application will access, if it does not already exist.
2. Create an SDE library for the database.
3. Prepare a SAMeDL source input file and compile it into the SDE library.
4. Write the Ada application program which uses the SAMeDL standard packages and the Ada definition/abstract modules generated by the SAMeDL compiler.
5. Compile and link the Ada application program.

4.1 Creating A Database

The initial creation and maintenance of a Oracle database is beyond the scope of SAMeDL. As described in the Oracle system administration manual [Oracle], the Database Administrator will create and maintain databases through the use of Oracle DBMS commands. Typical tasks would include:

- Create a Database
- Create the Database files(tables) and fields(columns) for the Database.
- Assign the appropriate permissions to the database to allow application connection through the Oracle HLI. This step includes adding login ids and users as necessary to the database via database administrator procedures.
- Set any default values and/or integrity constraints on table fields.

4.2 Creating An SDE Library

Once an appropriate database exists, you need to create an SDE library before you can compile SAMeDL source code. The SDE library will be used by the SAMeDL compiler to store information necessary for separate compilation and also to act as a repository for the interface files that are generated.

You create a new SDE library with the `sde.creatlib` command. This command optionally takes one argument which is the directory name for the library; if you do not specify an argument, then the library will be generated in your current working directory.

For example, to create an SDE library in the directory `/usr/same/example/samelib`, you would issue the following command:


```
%sde.creatlib /usr/same/example/samelib
```

In order to create the library, it is important that you have appropriate read/write privileges for the library directory.

4.3 Compiling A SAMeDL Source File

The next step is to prepare a SAMeDL source file (with the text editor of your choice) and compile it into the SDE Library.

Before you can use the compiler however, you must properly set the environment variable **ORACLE_HOME** to contain the path name to the Oracle RDBMS installation directory (e.g., */usr/oracle*).

Consider the following description which is assumed to be in the file **bank.sme**. This example contains three modules: the definition module **samplemod**, the schema module **recdb**, and the abstract module **recdml**. Furthermore, the example depends on the definition module **samedl_standard** which must have been previously compiled into your SAMeDL library.

```
--!reference samedl_standard
with samedl_standard; use samedl_standard;
definition module SampleMod is

    -- Member Information
    domain Member_Name is new SQL_CHAR Not Null (LENGTH => 30);
    domain SSN is new SQL_CHAR Not Null (LENGTH => 9);
    domain Age is new SQL_SMALLINT ( FIRST => 1, LAST => 199.0);

    enumeration SexEnum is (F, M);
    domain Sex is new SQL_ENUMERATION_AS_CHAR (
        ENUMERATION => SexEnum,
        WIDTH => 1,
        MAP =>(m=>'B', f=>'A'));

    domain Phone is new SQL_CHAR (LENGTH => 8);
    domain Street is new SQL_CHAR (LENGTH => 30);
    domain City is new SQL_CHAR (LENGTH => 15);

    domain County is new SQL_CHAR Not Null (LENGTH => 2);

    domain Club_Number is new SQL_SMALLINT Not Null;

    exception Record_Not_Found;

    enumeration FailType is (Not_Logged_In, SQL_Ok, SQL_Fail);

    status fetch_map named is_found uses Failtype is
        ( -999 .. -300 => SQL_Fail,
          -299, -298 => Not_Logged_In,
           0 => SQL_Ok,
          100 => raise samplemod.record_not_found);

end SampleMod;
```

```
with SampleMod; use SampleMod;
schema module RecDB is
```

```
    table Members is
        MemberName not null : Member_Name,
        MemberSSN not null : SSN,
        ClubNumber not null : Club_Number,
        MemberAge      : Age,
        MemberSex       : Sex,
        MemberPhone     : Phone,
        MemberStreet    : Street,
        MemberCity      : City,
        MemberCnty not null : County
    end Members;
```

```
end RecDB;
```

```
with SampleMod; use SampleMod;
extended abstract module RecDML is
    authorization RecDB
```

```
    record MemberRec is
        MemberName : Member_Name;
        MemberSSN  : SSN;
        ClubNumber : Club_Number;
        MemberAge  : Age;
        MemberSex  : Sex;
        MemberPhone : Phone;
        MemberStreet : Street;
        MemberCity : City;
        MemberCnty : County;
    end;
```

```
    procedure CommitWork is
        commit work;
```

```
    extended procedure connect_oracle_server is
        connect server 'test' 'test';
```

```
    procedure MemberInsert is
        insert into RecDB.Members
        from Row : MemberRec VALUES;
```

```
    cursor MemberSelect (Req_MemberSSN : SSN) for
        select MemberName,
               MemberSSN,
               ClubNumber,
               MemberAge,
               MemberSex,
               MemberPhone,
               MemberStreet,
               MemberCity,
               MemberCnty
        from RecDB.Members
        where RecDB.Members.MemberSSN = Req_MemberSSN;
    is
```

```
        procedure FetchIt is
            fetch into Row : MemberRec
            status Fetch_Map named Rec_Status;

        end MemberSelect;

    end RecDML;
```

The SAMeDL compiler is invoked with the command **samedl**. For example, to compile **bank.sme** into the SDE library created above, you should issue the following command:

```
%samedl -library /usr/same/example/samelib bank.sme
```

The **-library** qualifier is used to specify the name of an existing SDE library; this is optional, and if not given, the library will be assumed to exist in your current working directory. You must give the host filename of the SAMeDL input source file; this filename must end with the characters **.sme**. For more information on invoking the SAMeDL compiler, refer to Chapter 5 of this manual.

The SAMeDL compiler will generate interface files for each definition module (an Ada package specification/body pair) and each abstract module (an Ada package specification/body pair, a C with embedded SQL file, and an object code file). No interface files are generated for schema modules. All interface files will be placed in the **samedl.lib** directory contained within the library directory. Thus, for the sample compiler invocation above, you can find all interface files in the directory **/usr/same/example/samelib/samedl.lib**.

To determine what the names of the generated interface files for the modules **samplemod** and **recdb**, you can use the **sde.ls** command. For example:

```
%sde.ls -l /usr/same/example/samelib -i samplemod recdml

samplemod
  Interface Files:
    /usr/same/example/samelib/samedl.lib/P_2_.a (ADASPEC)
    /usr/same/example/samelib/samedl.lib/B_2_.a (ADABODY)

recdml
  Interface Files:
    /usr/same/example/samelib/samedl.lib/P_3_.a (ADASPEC)
    /usr/same/example/samelib/samedl.lib/B_3_.a (ADABODY)
    /usr/same/example/samelib/samedl.lib/E_1.ec (EMBEDDED)
    /usr/same/example/samelib/samedl.lib/E_1.o (OBJECTFILE)
```

For more information concerning the naming conventions used for SDE library files, see Section 3.3 of this document.

4.4 Creating An Ada Application Program

The Ada files produced by the SAMeDL compiler along with the SAMeDL standard packages provide an abstract Ada interface to the database which may be utilized by an Ada application program. So before you can build your application, you first need to compile these files into an appropriate Ada library that will be visible to your Ada application development library.

The SAMeDL standard packages are provided as part of SDE. To determine the location of these files at your site, please refer to the SDE installation notes or ask your system administrator.

To generate an "invoke" command file for compiling the Ada interface files contained in your SAMeDL library into your Ada library, you may use the `sde.mkscript` command. For example:

```
%sde.mkscript -l /usr/same/example/samelib -o my_script samplemod recdml
%more my_script
Compile (Source => "/usr/same/example/samelib/samedl.lib/P_1.a");
Compile (Source => "/usr/same/example/samelib/samedl.lib/B_1.a");
Compile (Source => "/usr/same/example/samelib/samedl.lib/P_2.a");
Compile (Source => "/usr/same/example/samelib/samedl.lib/B_2.a");
Compile (Source => "/usr/same/example/samelib/samedl.lib/P_3.a");
Compile (Source => "/usr/same/example/samelib/samedl.lib/B_3.a");
```

In this example, the `sde.mkscript` command indicates that 3 sets of Ada package spec/body pairs need to be compiled, even though the initial compilation of the file containing `samplemod` and `recdml` generated only 2 Ada packages spec/body pairs. The reason for this discrepancy is that the definitional module `samplemod` references the previously-compiled module `samedl_standard`, which contains the definitions of the base domains `SQL_CHAR`, `SQL_INT`, etc. The reference to `samedl_standard` is achieved via a reference directive. For more information on compiler directives, see Section 5.4 of this manual.

To generate a C-language archive including all of the C object files pertinent to your Ada application you may use the `sde.creatar` command. For example:

```
%sde.creatar c_recddl.a recdml
```

The command given above will create a C-language archive named `c_recddl.a` which contains the C object code necessary to link the Ada-Oracle interface generated by the SAMeDL abstract module `recdml` to an Ada application, such as the one presented below.

Using the bank example presented above, suppose that you need a utility that will allow bank tellers access to profile information for a customer. You could accomplish this with the following Ada program:

```
with TEXT_IO;
use TEXT_IO;
with SAMPLEMOD;
with RECDML;
procedure MAIN is

  -- User I/O information
  IN_BUFFER : STRING(1 .. 80);
  LAST      : NATURAL;
  OPT       : INTEGER;

  -- Members Row Record
  ROW       : RECDML.MEMBERREC;
  IROW      : RECDML.MEMBERREC;

  procedure DO_INSERT is
  begin
    PUT_LINE(**** Function to Insert rows ****);
    NEW_LINE;
```

```
loop
  IN_BUFFER := (others => ' ');
  PUT("Enter Member SSN (9 char max) or -1 for MENU> ");
  GET_LINE(IN_BUFFER, LAST);
  NEW_LINE;
  exit when (IN_BUFFER(1 .. LAST) = "-1");

  IROW.MEMBERSSN := SAMPLEMOD.SSN_NOT_NULL(IN_BUFFER(1 .. 9));

  IN_BUFFER := (others => ' ');
  PUT("Enter Member Name (30 char max)> ");
  GET_LINE(IN_BUFFER, LAST);
  NEW_LINE;
  IROW.MEMBERNAME :=
    SAMPLEMOD.MEMBER_NAME_NOT_NULL(IN_BUFFER(1 .. 30));

  IN_BUFFER := (others => ' ');
  PUT("Enter Club Number (Smallint)> ");
  GET_LINE(IN_BUFFER, LAST);
  NEW_LINE;
  IROW.CLUBNUMBER :=
    SAMPLEMOD.CLUB_NUMBER_NOT_NULL'VALUE(
      IN_BUFFER(1 .. LAST));

  IN_BUFFER := (others => ' ');
  PUT("Enter Member Age (Smallint) or \\ for NULL> ");
  GET_LINE(IN_BUFFER, LAST);
  NEW_LINE;
  if (IN_BUFFER(1 .. 2) = "\\") then
    SAMPLEMOD.AGE_OPS.ASSIGN(IROW.MEMBERAGE,
      SAMPLEMOD.NULL_SQL_SMALLINT);
  else
    SAMPLEMOD.AGE_OPS.ASSIGN(IROW.MEMBERAGE,
      SAMPLEMOD.AGE_OPS.WITH_NULL(
        SAMPLEMOD.AGE_NOT_NULL'VALUE(
          IN_BUFFER(1 .. LAST))));
  end if;

  IN_BUFFER := (others => ' ');
  PUT("Enter Member Sex (M/F) or \\ for NULL> ");
  GET_LINE(IN_BUFFER, LAST);
  NEW_LINE;
  if (IN_BUFFER(1 .. 2) = "\\") then
    SAMPLEMOD.ASSIGN(IROW.MEMBERSEX,
      SAMPLEMOD.NULL_SQL_ENUMERATION);
  else
    SAMPLEMOD.ASSIGN(IROW.MEMBERSEX,
      SAMPLEMOD.WITH_NULL(
        SAMPLEMOD.SEX_NOT_NULL'VALUE(
          IN_BUFFER(1 .. LAST))));
  end if;

  IN_BUFFER := (others => ' ');
  PUT("Enter Member Phone (8 chars) or \\ for NULL> ");
  GET_LINE(IN_BUFFER, LAST);
  NEW_LINE;
  if (IN_BUFFER(1 .. 2) = "\\") then
```

```

        SAMPLEMOD.ASSIGN(IROW.MEMBERPHONE,
            SAMPLEMOD.NULL_SQL_CHAR);
    else
        SAMPLEMOD.ASSIGN(IROW.MEMBERPHONE,
            SAMPLEMOD.PHONE_OPS.WITH_NULL(
                SAMPLEMOD.PHONE_NOT_NULL(IN_BUFFER(1 .. 8))));
    end if;

    IN_BUFFER := (others => ' ');
    PUT("Enter Member Street (30 char max) or \\ for NULL> ");
    GET_LINE(IN_BUFFER, LAST);
    NEW_LINE;
    if (IN_BUFFER(1 .. 2) = "\\") then
        SAMPLEMOD.ASSIGN(IROW.MEMBERSTREET,
            SAMPLEMOD.NULL_SQL_CHAR);
    else
        SAMPLEMOD.ASSIGN(IROW.MEMBERSTREET,
            SAMPLEMOD.STREET_OPS.WITH_NULL(
                SAMPLEMOD.STREET_NOT_NULL(IN_BUFFER(1 .. 30))));
    end if;

    IN_BUFFER := (others => ' ');
    PUT("Enter Member City (15 char max) or \\ for NULL> ");
    GET_LINE(IN_BUFFER, LAST);
    NEW_LINE;
    if (IN_BUFFER(1 .. 2) = "\\") then
        SAMPLEMOD.ASSIGN(IROW.MEMBERCITY,
            SAMPLEMOD.NULL_SQL_CHAR);
    else
        SAMPLEMOD.ASSIGN(IROW.MEMBERCITY,
            SAMPLEMOD.CITY_OPS.WITH_NULL(
                SAMPLEMOD.CITY_NOT_NULL(IN_BUFFER(1 .. 15))));
    end if;

    IN_BUFFER := (others => ' ');
    PUT("Enter Member Cnty (2 char max)> ");
    GET_LINE(IN_BUFFER, LAST);
    NEW_LINE;
    IROW.MEMBERCNTY :=
        SAMPLEMOD.COUNTY_NOT_NULL(IN_BUFFER(1 .. 2));

    RECDML.MEMBERINSERT(IROW);
    RECDML.COMMITWORK;
end loop;

exception
    when others =>
        PUT_LINE("**** Error: could not do Insert ****");
end DO_INSERT;

procedure DO_SELECT is
    STATUS : SAMPLEMOD.FAILTYPE;
begin
    PUT_LINE("**** Function to Select rows ****");
    NEW_LINE;
    loop
        IN_BUFFER := (others => ' ');
        PUT("Enter Member SSN (9) or -1 for MENU> ");

```

```
GET_LINE(IN_BUFFER, LAST);
NEW_LINE;
exit when (IN_BUFFER(1 .. LAST) = "-1");
RECDML.MEMBERSELECT.OPEN(SAMPLEMOD.SSN_NOT_NULL(
    IN_BUFFER(1 .. 9)));

begin
    loop
        RECDML.MEMBERSELECT.FETCHIT(ROW, STATUS);

        PUT_LINE("NAME: " & STRING(ROW.MEMBERNAME) & "      "&
            "SSN: " & STRING(ROW.MEMBERSSN) & "      "&
            "CLUB: " &
            SAMPLEMOD.CLUB_NUMBER_NOT_NULL' IMAGE(
                ROW.CLUBNUMBER));

        PUT("AGE: ");
        if not (SAMPLEMOD.IS_NULL(ROW.MEMBERAGE)) then
            PUT(SAMPLEMOD.AGE_NOT_NULL' IMAGE(
                SAMPLEMOD.AGE_OPS.WITHOUT_NULL(
                    ROW.MEMBERAGE)));
        end if;
        SET_COL(13);

        PUT("SEX: ");
        if not (SAMPLEMOD.IS_NULL(ROW.MEMBERSEX)) then
            PUT(SAMPLEMOD.SEX_NOT_NULL' IMAGE(
                SAMPLEMOD.WITHOUT_NULL(ROW.MEMBERSEX)));
        end if;
        NEW_LINE;

        PUT("PHONE: ");
        if not (SAMPLEMOD.IS_NULL(ROW.MEMBERPHONE)) then
            PUT(STRING(SAMPLEMOD.PHONE_OPS.WITHOUT_NULL(
                ROW.MEMBERPHONE)));
        end if;
        NEW_LINE;

        PUT("STREET: ");
        if not (SAMPLEMOD.IS_NULL(ROW.MEMBERSTREET)) then
            PUT(STRING(SAMPLEMOD.STREET_OPS.WITHOUT_NULL(
                ROW.MEMBERSTREET)));
        end if;
        NEW_LINE;

        PUT("CITY: ");
        if not (SAMPLEMOD.IS_NULL(ROW.MEMBERCITY)) then
            PUT(STRING(SAMPLEMOD.CITY_OPS.WITHOUT_NULL(
                ROW.MEMBERCITY)));
        end if;
        SET_COL(26);

        PUT_LINE("COUNTY: " & STRING(ROW.MEMBERCNTY));
        NEW_LINE;
        PUT_LINE("*****");
        NEW_LINE;
    end loop;
```

```

        exception
            when others =>
                PUT_LINE("No more records found!");
                NEW_LINE;
        end;

        RECDML.MEMBERSELECT.CLOSE;
        RECDML.COMMITWORK;

    end loop;

    exception
        when others =>      -- Couldn't find request
            PUT_LINE("**** Error: could not do Select ****");

    end DO_SELECT;

begin
    RECDML.CONNECT_ORACLE_SERVER;
    loop
        PUT_LINE("**** Option Menu ****");
        PUT_LINE("    0 - Quit");
        PUT_LINE("    1 - Insert");
        PUT_LINE("    2 - Select");
        PUT("Option? > ");
        GET_LINE(IN_BUFFER, LAST);
        NEW_LINE;
        OPT := INTEGER'VALUE(IN_BUFFER(1 .. LAST));

        case OPT is
            when 0 =>
                exit;
            when 1 =>
                DO_INSERT;
            when 2 =>
                DO_SELECT;
            when others =>
                PUT_LINE("Illegal Choice: " & IN_BUFFER(1 .. LAST));
        end case;
        NEW_LINE(2);
    end loop;
end MAIN;

```

Assuming that your Alsys Ada development library is in `/usr/same/example/adalib` and that the above Ada program is in the file `getprof.a`, you can compile and link the program by performing the following steps, which use the file `myscript` and the archive file `c_recdml.a` described earlier in this section:

- Compile the SAMeDL Standard Packages into your Ada library (this step may be omitted if visibility to the SAMeDL Standard Packages has been gained in another way).


```

      %$SDEPATH/comp_std_pkgs /usr/same/example/adalib
      
```
- Compile the code generated by the SAMeDL compiler into your library, using the script generated by the `sde.mkscript` command.


```
%ada
> default.compile(library=>/usr/same/example/adalib)
> invoke (file=>myscript)
> quit
```

- Compile your application into the Ada library.

```
%ada
> default.compile(library=>/usr/same/example/adalib)
> compile (source=>getprof.ada)
> quit
```

- Generate the executable using the Alsys BIND command. When issuing the BIND command, include the following arguments in addition to providing values for the required PROGRAM and LIBRARY parameters [AdaRef]:

(a) MODULES => \$ORACLE_HOME/rdbms/lib/osntab.o

(b) SEARCH => the archive file generated by the call to sde.creatar (for this example, c_recdml.a),
\$ORACLE_HOME/rdbms/lib/libsqlnet.a,
\$ORACLE_HOME/rdbms/lib/libsql.a,
\$ORACLE_HOME/rdbms/lib/libora.a

- The order of arguments for the MODULES parameter is not significant, but the order of arguments for the SEARCH parameter is significant. If your application program requires additional external modules, you may have to reorder the list of external modules before all references can be adequately resolved. Consult the Alsys Ada User's Manual for further information.

In the example above, the Unix C-shell (csh) script `comp_std_pkgs` provided with SDE contains Alsys Ada Compiler commands and is described in Section 5.3 of this document (SDEPATH is an environment variable which has been set to the path name for the SDE installation directory).

Chapter 5 Building Ada/SQL Interfaces With SAMeDL

5.1 Overview Of The SAMeDL Compiler

The SAMeDL compiler is used to generate interface files representing an Ada/SQL interface for your Ada applications. These interface files consist of one or more files containing Ada packages representing the Ada interface:

- Each definition module defined in the source input will have an Ada package specification and a corresponding Ada package body generated.
- Each abstract module defined in the source input will have an Ada package specification and a corresponding Ada package body generated.

In addition, for each abstract module a corresponding *concrete* module will be generated. This file takes the form of C code with embedded Oracle SQL statements. Procedures declared within the file are called by procedures within the abstract module's Ada package body in order that direct interaction with the database can be handled. Each such file will be preprocessed by the Oracle SQL C preprocessor and the resulting output will be compiled by the C compiler resulting in a corresponding object code file (a .o file).

The SAMeDL compiler operates within the context of an SDE library. The library maintains dependency information and other data used by the compiler to perform separate compilation. In addition, the SDE library acts as a repository for all interface files generated by the SAMeDL compiler.

5.2 SAMeDL Compiler Invocation

The SAMeDL compiler is invoked with the command **samedl**. It accepts a series of options and a single file name as input arguments. Option keywords are not case sensitive and may be truncated as long as the resulting abbreviation is unambiguous.

Syntax

samedl [*options*] *source_file*

Options

-library <i>pathname</i>	Operate in the SDE library <i>pathname</i> . If not specified, will default to current working directory
-list	Generate an interleaved listing file
-syntax	Check the syntax of the input file without generating any output files.

samedl executes the SAMeDL compiler and compiles the named SAMeDL source file into the SDE library directory specified by *pathname*; if *pathname* is not specified, then it will default to

the current working directory. Note that the SDE library must already have been created (via the `sde.creatlib` command). The SAMeDL source file name must end with the suffix `.sme`.

The listing option, when specified, directs the compiler to produce an interleaved listing file. The listing file will be named `<x>.lis` where `<x>` is the base name of the input source file (for example, a source file named `xyz.sme` will result in a listing file being named `xyz.lis`). Compiler diagnostic messages will always be written to standard output, regardless of whether or not `-list` is in effect.

The syntax option, when specified, causes the SAMeDL compiler to act as a SAMeDL syntax checker, generating error messages for syntax and some semantic errors, but no code.

The SAMeDL compiler will generate interface files for each definition module (an Ada package specification/body pair) and each abstract module (an Ada package specification/body pair, a C with embedded SQL file, and an object code file). No interface files are generated for schema modules. All interface files will be placed in the `samedl.lib` directory contained within the library directory. For the naming conventions used for interface files, please refer to Section 3.3 in this manual.

As an example, take the following:

```
%samedl -lib /usr/same/example/samelib -list example.sme
```

This will compile the SAMeDL description file `example.sme` into the library `/usr/same/example/samelib` and create an interleaved listing file named `example.lis` in the current directory. All generated interface files will be placed in the directory `/usr/same/example/samelib/samedl.lib`.

Before invoking the SAMeDL Compiler, users should be sure to check that SAMeDL packages required via reference directives have already been compiled into the SAMeDL library. In particular, a typical SAMeDL code file will include reference directives for the SAMeDL definitional modules `SAMeDL Standard` and `SAMeDL System`, found in the files `$$DEPATH/STD PKGS/samedl_std.sme` and `$$DEPATH/STD PKGS/samedl_sys.sme`. These packages contain definitions for system limits and predefined base domains. Users who tend to frequently use the predefined base domains should get into the habit of compiling these files into their SAMeDL libraries at library creation time.

Before you can use the compiler however, you must properly set the environment variable `ORACLE_HOME` to contain the path name to the Oracle RDBMS installation directory (e.g., `/usr/oracle`).

5.3 Using the Compiler-Generated Interface

In order to use the SAMeDL compiler generated Ada/SQL interface, the target Ada application must be linked with the SAMeDL generated Ada files and object code files, a set of SAMeDL standard packages (see Section 3.5), a set of Oracle Libraries, and an Oracle object file. To facilitate the final steps in building the Ada target application, SDE provides you with a Unix C-shell script that contains Alsys Ada Compiler commands. This script can be found in the SDE installation directory and used as an example of how to compile the SAMeDL Standard Packages your application requires using the Alsys Ada Compiler. It is called `comp_std_pkgs`.

The first step in compiling and linking your application is to make the SAMeDL standard packages visible to your Alsys Ada application library. This can be done by using the **comp_std_pkgs** script file found in the SDE installation directory. You may invoke the **comp_std_pkgs** by issuing the following command:

```
%comp_std_pkgs libpath
```

where **libpath** is the pathname to the Ada library that the SAMeDL standard packages are to be compiled into. This script will compile all of the SAMeDL standard packages into your Ada library. This step needs to be performed once per library, unless the SAMeDL standard packages have already been made visible to the Ada library in some other way.

Once the standard packages have been compiled into the Ada library, the SAMeDL-generated Ada packages should be compiled into the library. The SDE command **sde.mkscript** can be used to generate a script file for performing this compilation. Refer to section 8.6 of this document for instructions and examples.

After the SAMeDL interface code has been compiled into your library, you may use the Alsys **Compile** command to compile your application into the library. Once this step has been completed, you are ready to prepare for linking the Ada-Oracle executable.

There are five files which must be linked with your application in order to produce a valid executable. Three of the files, namely **\$ORACLE_HOME/rdbms/lib/libsqlnet.a**, **\$ORACLE_HOME/rdbms/lib/libsql.a** and **\$ORACLE_HOME/lib/libora.a** are Oracle libraries. One of the files, **\$ORACLE_HOME/rdbms/lib/osntab.o**, is an Oracle object file. And the fifth file is an archive of the pertinent C object code generated by the SAMeDL compiler and stored in the SAMeDL library. This last file is created from the SAMeDL library information using the command **sde.creatar**, described in section 8.3 of this document.

Your Ada application can be linked easily by following these simple instructions:

- Generate the executable using the Alsys **BIND** command. When issuing the **BIND** command, include the following arguments in addition to providing values for the required **PROGRAM** and **LIBRARY** parameters [AdaRef]:
 - (a) **MODULES => \$ORACLE_HOME/rdbms/lib/osntab.o**
 - (b) **SEARCH => the archive file generated by the call to sde.creatar (for this example, c_recddl.a),
\$ORACLE_HOME/rdbms/lib/libsqlnet.a,
\$ORACLE_HOME/rdbms/lib/libsql.a,
\$ORACLE_HOME/lib/libora.a**
- The order of arguments for the **MODULES** parameter is not significant, but the order of arguments for the **SEARCH** parameter is significant. If your application program requires additional external modules, you may have to reorder the list of external modules before all references can be adequately resolved. Consult the Alsys Ada User's Manual for further information.

5.4 Compiler Directives

Compiler directives are embedded in SAMeDL source files and are used to indicate special directions to the compiler outside of the SAMeDL syntax and semantics. The general form of any directive is:

`--ldirective_name parameter_list`

In order for a directive to be recognized, it is important that no *white space* (i.e., spaces, tabs, etc.) appear between any of the *dashes* (-), the *bang* (!), and the *directive_name* keyword.

Each directive will be given in its general form, followed by a definition of each term of the directive, and a description of its use.

5.4.1 Reference Directive

The reference directive allows you flexibility of separate compilation by permitting visibility of externally declared modules that have been previously compiled. This directive(s) must appear immediately before the first context clause of a SAMeDL module.

The compiler processes the reference directive by reading the referenced module from the SDE library currently in context and importing the appropriate symbol information for the referenced module. Once a reference directive is used for a particular module, then any module appearing textually after the reference directive may refer to the contents of the referenced module.

Typical use for the reference directive is to gain visibility to the SAMeDL packages SAMeDL_Standard and SAMeDL_System, which contain the definitional modules for the predefined base domains and the system limits.

The form of the reference directive is as follows:

`--lreference module_name`

The **reference** keyword must begin immediately following the **!** and the entire word must be included. The keyword is case-insensitive. `module_name` must reference the name of a SAMeDL module that has been previously compiled into the SDE library.

Note: This directive must be placed before the context clauses of a module declaration; placing it between the start of a module declaration and the corresponding **END** will cause a fatal error. Also, this directive will not compile the referenced module. Any module that needs to be compiled or re-compiled, needs to be done so separately.

As an example, assume the following definitional and schema modules have been previously compiled.

```
DEFINITION MODULE Bank_Def IS
  DOMAIN Customer_name_domain IS NEW SQL_CHAR(length => 50);
  .
  .
  .
END Bank_Def;
```

```
WITH Bank_Def;  
USE Bank_Def;  
SCHEMA MODULE BankDB IS  
  
.  
.  
.  
  
END BankDB;
```

Then the following Abstract module would have full visibility to both modules using the reference directive:

```
--!Reference bank_def  
--!Reference bankdb  
WITH Bank_Def;  
USE Bank_Def;  
ABSTRACT MODULE Bank_Actions IS  
    AUTHORIZATION BankDB  
  
.  
.  
.  
  
END Bank_actions;
```

5.4.2 Owner Directive

The owner directive enables you to specify the Oracle *owner* of a particular set of database tables. The owner directive must precede a schema module declaration and affects that schema module in the following way: the owner_name specified in the directive is considered to be the name of the Oracle owner of the database tables defined in the schema module. Each owner directive applies only to the next schema module declaration in the SAMeDL source code. If no owner is specified for a particular schema module, then the owner is assumed (by Oracle) to be the user-name of the Oracle account which the application connected to via the SAMeDL connect server command.

The format of the owner directive is as follows:

```
--!owner owner_name
```

The keyword **owner** must begin immediately following the **!** and the entire word must be included. The keyword is case-insensitive.

Note: This directive must be placed outside of any module declaration; placing it between the start of a module declaration and the corresponding **END** will cause a fatal error. The most logical place to put the directive is directly before a schema module declaration, as shown below.

As an example, use of the Owner Directive, as exhibited below, would cause the resulting Ada-Oracle application to access the table **myuserid.Cust**, owned by user **myuserid**:

```
--!owner myuserid
```

```
WITH Bank_Def;
USE Bank_Def;
SCHEMA MODULE BankDB IS
  TABLE Cust IS -- Basic customer information
    Name      : Customer_name_domain,
    SSN       : SSN_domain,
    Street_addr : Addr_domain,
    City_addr  : Addr_domain,
    State_addr  : State_domain
  END Cust;
END BankDB;

WITH Bank_Def;
USE Bank_Def;
ABSTRACT MODULE Bank_Actions IS
  AUTHORIZATION BankDB

  PROCEDURE Get_customer_profile ( SSN_IN : SSN_domain ) IS
    SELECT *
      INTO Customer_profile : customer_record
     FROM BankDB.Cust
    WHERE SSN = SSN_in;

END Bank_actions;
```

Note: Successful use of the Owner Directive requires that the resulting Ada-DBMS application be run from an account which has been granted the appropriate privileges for all referenced tables. Refer to the Oracle user's guides [Oracle] for more information on privileges and owners.

Chapter 6 Implementation Dependent Features

This chapter describes SAMEDL features which are dependent on the Oracle implementation. Section 6.1 describes features which are included as part of the SAMEDL language ([LRM]) but not supported due to limitations imposed by Oracle. Section 6.2 details features which are not included as part of the SAMEDL but are provided as extensions for the implementation either because of necessity or convenience. Finally, Section 6.3 includes some solutions to system errors that are commonly encountered.

6.1 SAMEDL Language Limitations Under Oracle

Because of limitations imposed by Oracle, use of the following features described in the SAMEDL Language Reference Manual ([LRM]) will produce errors (all references below are made with respect to [LRM]):

1. **WHERE Clauses** - The SAMEDL language, in accordance with the ANSI SQL Standard, allows input references to be part of value expressions, regardless of whether or not the input reference is to a null-bearing parameter. However, Oracle does not allow null-bearing input references to be part of where clauses. The SAMEDL compiler will issue an error if a null-bearing parameter is used in a place where Oracle does not allow it.

6.2 SAMEDL Extensions For Oracle

This section details features which are included as part of SAMEDL as implementation-specific extensions either because of necessity or convenience. They include the following statements:

Connect Server Statement

The connect server statement is an extended statement. It's grammar consists of the following productions:

```
connect_server_statement ::= connect server user_id password
                           [ using database_name ] ;

user_id ::= input_param_ref |
            character_literal |
            constant_reference

password ::= input_param_ref |
            character_literal |
            constant_reference

database_name ::= input_param_ref |
                  character_literal |
                  constant_reference
```

The Connect Server Statement connects the application to the Oracle server named *database_name* as the user named by *user_id* with the password given by *password*. The input parameters representing the user id, password, and database name must have been declared with a SAMEDL domain whose DBMS-type is **character**. If the *using* clause is not specified, then

database_name will default to the user's default table space (see Appendix D, [OracleESQL]). All subsequent transactions are performed on the connected database. No data access can be performed until the application has successfully connected to the server.

Because the Connect Server statement is an extended statement, its containing procedure and abstract module must be marked as extended.

Definitional Module Bodies

The SAMeDL Compiler generates a package body for each definitional module. This practice differs from the recommendation of the SAMeDL LRM, but is maintained in order to decrease code size and functional redundancy.

The package body for each definitional module is empty unless the definitional module contains a domain declaration of data class **enumeration** possessing a user-defined database mapping as a value for the predefined parameter **MAP**. For each declaration of this type, a function to perform conversion from the domain type to the underlying database type is provided. A function to convert from the database type to the domain type is also provided. Without these globally accessible functions, a large amount of code would have to be reproduced frequently in the Abstract Module's package body in order to perform data conversions.

These functions can be accessed by the SAMeDL application, but are primarily designed for use by the SAMeDL compiler back-end to generate package bodies for Abstract Modules.

6.3 Troubleshooting Common System Errors

The following list includes some helpful techniques for configuring the SAMeDL environment that will reduce your chances of getting some common system errors.

1. **Increase the MAXUMEM Interactive Unix kernel parameter** - If you get an Oracle error message when running your application that indicates that your application process does not have enough memory available to run successfully, then you should increase the MAXUMEM kernel parameter. Consult your Unix system administrator for instructions.
2. **Increase the Interactive Unix ULIMIT kernel parameter** - If you get an Alsys error message during the LISTING phase of compiling your application, then you might try raising the ULIMIT kernel parameter to increase the file size limit. Consult your Unix system administrator for instructions.

Chapter 7 Tool Limitations

This chapter lists limitations of SDE.

7.1 SAMeDL Compiler Limitations

The following limitations are imposed by the SAMeDL compiler:

- The maximum number of characters allowed in a source line is 255.
- The compiler will not delete any files from an SDE library; the **sde.purge** command must be used to clean the library of any out of date or temporary files.
- The maximum length of an Error Message that can be printed by the `Process_Database_Error` routine is 132 characters.
- If extremely long names are used in the SAMeDL source code, it is possible that the compiler could attempt to generate output with lines that exceed the Unix line length limit. The SAMeDL compiler will issue a warning if excessive name length results in an output problem.
- The value range for types `Smallint` and `Indicator_Type` is -32768 .. 32767.
- The value range for types `Int` and `Sqlcode_Type` is -2147483648 .. 2147483647.
- The value range for type `Real` is the range for Alsys' `LONG_FLOAT` type.

In addition, because Ada source is generated by the SAMeDL compiler, all restrictions and semantics as outlined in [Ada] and [AdaRef] must be followed. Although these limits are not explicitly checked by the SAMeDL compiler, they do indirectly affect the structure of what normally would be legal SAMeDL code.

7.2 SDE Module Manager Limitations

The following limitations are imposed by the SDE Module Manager:

1. The SDE commands (with the exception of **sde.purge**) will not delete any files from an SDE library; the **sde.purge** command must be used to clean the library of any out of date or temporary files.
2. After executing the **sde.purge** command, you may not restore the library to its prior state.

Chapter 8 SDE Command Reference Manual Pages

This chapter contains a reference guide for each of the commands in SDE. The commands available to you are:

samedl	invoke the SAMeDL compiler
sde.cleanlib	reinitialize an SDE library
sde.creatar	create a library archive file for compiled concrete modules
sde.creatlib	create an SDE library
sde.ls	list compiled SAMeDL modules
sde.mkscript	generate an Ada compilation script file for an interface file
sde.purge	remove out of date files from an SDE library
sde.rm	remove a SAMeDL module from an SDE library
sde.rmlib	remove an SDE library

8.1 samedl

Command

samedl - invoke the SAMeDL compiler

Syntax

samedl [*options*] *source_file*

Options

- | | |
|---------------------------------|--|
| -library <i>pathname</i> | Operate in the SDE library <i>pathname</i> . If not specified, will default to current working directory |
| -list | Generate an interleaved listing file |
| -syntax | Check the syntax of the input file without generating any output files. |

Description

samedl executes the SAMeDL compiler and compiles the named SAMeDL source file into the SDE library directory specified by *pathname*; if *pathname* is not specified, then it will default to the current working directory. The SAMeDL source file name must end with the suffix *.sme*.

The listing option, when specified, directs the compiler to produce an interleaved listing file. The listing file will be named *<x>.lis* where *<x>* is the base name of the input source file (for example, a source file named *xyz.sme* will result in a listing file being named *xyz.lis*). Compiler diagnostic messages will always be written to standard output, regardless of whether or not **-list** is in effect.

The SAMeDL compiler will generate interface files for each definition module (in the form of an Ada package specification/body pair) and each abstract module (in the form of a layered interface consisting of an Ada package specification/body pair and an object code file generated from a C with embedded SQL file). No interface files are generated for schema modules. All interface files will be placed in the SAMeDL library contained within the library directory.

Before using the compiler, the environment variable **ORACLE_HOME** must be properly set to contain the path name to the ORACLE RDBMS installation directory (e.g., */usr/oracle*).

Module Type	File Name	Description
Definitional Module	D_XXXXX.sme	Text file containing SAMeDL source code representing the definitional module
	P_XXXXX_a	Generated Ada package specification file
	B_XXXXX.a	Generated Ada package body file
Schema Module	S_XXXXX.sme	Text file containing SAMeDL source code representing the schema module
Abstract Module	A_XXXXX.sme	Text file containing SAMeDL source code representing the abstract module
	P_XXXXX_a	Generated Ada package specification file
	B_XXXXX.a	Generated Ada package body file
	E_XXXXX.ec	Generated C w/ embedded SQL (C/ESQL) file
	E_XXXXX.o	Object code for the expanded/compiled C/ESQL file

where XXXXX denotes a unique integer.

Diagnostics

The diagnostics produced by the SAMeDL compiler are intended to be self-explanatory.

8.2 sde.cleanlib

Command

sde.cleanlib - reinitialize a SDE library

Syntax

sde.cleanlib [*pathname*]

Description

sde.cleanlib will empty an existing SDE library of all compilation information. The command will re-initialize the **names.dbc** and **samedl.dat** files and remove the remaining contents of the **samedl.lib** directory from the directory specified by *pathname*; if *pathname* is not specified, then it will default to the current working directory.

Examples

The following sequence of commands cleans and re-initializes the library contained in the directory **/home/samedl**.

```
%cd /home/samedl
%sde.cleanlib
```

The following command does the same thing:

```
%sde.cleanlib /home/samedl
```

Diagnostics

An error is reported and no action is taken if *pathname* does not specify a valid, unlocked SDE library.

8.3 sde.creatar

Command

sde.creatar - create a library archive file for compiled concrete modules

Syntax

sde.creatar [*options*] *archive_name module_name ...*

Options

-library <i>pathname</i>	Operate in the SDE library <i>pathname</i> . If not specified, will default to current working directory
---------------------------------	--

Description

For each SAMeDL abstract module specified by *module_name*, **sde.creatar** will add (or replace) the object code file representing the related concrete module into the library archive file denoted by *archive_name*. The library archive file may already exist, or in the event that it does not exist, a new one will be created. **sde.creatar** is analogous to the following UNIX command:

```
%ar r archive_name c_module_name1.o ...
```

See **ar(1)** in the UNIX Programmer's Manual.

Examples

The following example adds the concrete modules associated with the SAMeDL abstract modules **abs1** and **abs2** (assume they are **E_1.o** and **E_2.o** respectively) from the library **/usr/home/jdoe/my_lib** to the archive file **my_archive** in the current working directory.

```
%sde.creatar -lib /usr/home/jdoe/my_lib my_archive abs1 abs2
```

Assuming that **my_archive** was previously empty or did not exist, then issuing the UNIX command **ar** with the table of contents (**t**) option will yield the following results:

```
%ar t ./my_archive
E_1.o
E_2.o
```

Diagnostics

An error is reported and no action is taken if *module_name* is not an abstract module or does not exist in the library, or if the library is not valid or is locked.

8.4 sde.creatlib

Command

sde.creatlib - create an SDE library

Syntax

sde.creatlib [*pathname*]

Description

sde.creatlib creates and initializes a new SDE library. It creates a directory named **samedl.lib** for the library in the directory specified by *pathname*. If *pathname* is not given, the current working directory is the default.

The command creates the files **samedl.dat** and **names.dbe** in the **samedl.lib** directory and sets the their information fields to an initial state.

Examples

The following sequence of commands creates a new SDE module manager library in the directory **/home/samedl**.

```
%cd /home/samedl
%sde.creatlib
```

The following command does the same thing:

```
%sde.creatlib /home/samedl
```

Diagnostics

An error is generated and no action is taken if *pathname* is not an existing directory or if the directory already contains an SDE library.

8.5

Command

sde.ls - list compiled SAMeDL modules

Syntax

sde.ls [*options*] [*module_name*] ...

Options

-ada_only	List only generated Ada interface files
-interface	List all generated interface files
-library <i>pathname</i>	Operate in the SDE library <i>pathname</i> . If not specified, will default to current working directory
-verbose	List file, file type, library entry date, source file name, and library file name.

Description

sde.ls provides a list of the SAMeDL modules compiled in the specified SDE library denoted by *pathname* (or the current working directory if *pathname* is not given). Options are provided to give more or less extensive information.

Specifying one or more module names gives information only on those modules; otherwise information for all modules in the library will be listed.

The options **-ada_only** and **-interface** are mutually exclusive. If both are specified, then **-interface** will be in effect.

Examples

The following command lists all (verbose) information for the modules **abs1** and **abs2** and their generated interface files from the library in the current working directory.

```
%sde.ls -v -i abs1 abs2

abs1
Unit Kind: ABSMODULE
Source File: abs1.sme
Library File: ./samedl.lib/A_1.sme
Time Entered: Feb 24 1992 11:59
Interface Files:
  ./samedl.lib/P_2_.a (ADASPEC)
  ./samedl.lib/B_2_.a (ADABODY)
  ./samedl.lib/E_1.ec (EMBEDDEDC)
  ./samedl.lib/E_1.o (OBJECTFILE)
```

abs2

Unit Kind: ABSMODULE

Source File: abs2.sme

Library File: ./samedl.lib/A_2.sme

Time Entered: Feb 24 1992 12:00

Interface Files:

./samedl.lib/P_3_.a (ADASPEC)

./samedl.lib/B_3_.a (ADABODY)

./samedl.lib/E_2.ec (EMBEDDED)

./samedl.lib/E_2.o (OBJECTFILE)

Diagnostics

An error is reported and no action is taken if *module_name* does not exist in the library, or if the library is not valid or is locked.

8.6 sde.mkscript

Command

sde.mkscript - generate an Ada compilation script file for an interface file

Syntax

sde.mkscript [*options*] *module_name* ...

Options

-library <i>pathname</i>	Operate in the SDE library <i>pathname</i> . If not specified, will default to current working directory
-output <i>filename</i>	Place the generated script template into <i>filename</i>

Description

sde.mkscript will create a template for performing the Ada compilation of the generated Ada files (and the units they depend on) for the definitional or abstract module(s) specified.

Examples

Suppose in the library `/usr/home/jdoe/my_lib` you have compiled the abstract module `my_abs` which depends (WITHs) the schema module `my_sch` and the definitional module `my_def`; `my_sch` depends only on `my_def` and `my_def` depends on no modules. Performing an `sde.ls` command gives the following information:

```
%sde.ls -v -a -l /usr/home/jdoe/my_lib my_abs my_def my_sch
```

```
my_abs
```

```
Unit Kind: ABSMODULE
Source File: input.sme
Library File: /usr/home/jdoe/my_lib/samed1.lib/A_1.sme
Time Entered: Feb 24 1992 11:59
Interface Files:
  /usr/home/jdoe/my_lib/samed1.lib/P_2.a (ADASPEC)
  /usr/home/jdoe/my_lib/samed1.lib/B_2.a (ADABODY)
```

```
my_def
```

```
Unit Kind: DEFMODULE
Source File: input.sme
Library File: /usr/home/jdoe/my_lib/samed1.lib/D_1.sme
Time Entered: Feb 24 1992 11:59
Interface Files:
  /usr/home/jdoe/my_lib/samed1.lib/P_1.a (ADASPEC)
  /usr/home/jdoe/my_lib/samed1.lib/B_1.a (ADABODY)
```

```
my_sch
  Unit Kind: SCHEMAMODULE
  Source File: input.sme
  Library File: /usr/home/jdoe/my_lib/samed1.lib/S_1.sme
  Time Entered: Feb 24 1992 11:59
```

You may issue a **sde.mkscript** command to generate an Ada compilation template for compiling the Ada interface files associated with **my_abs** as follows:

```
%sde.mkscript -l /usr/home/jdoe/my_lib -o my_script my_abs
%more my_script
Compile (Source => "/usr/same/example/samelib/samed1.lib/P_1.a");
Compile (Source => "/usr/same/example/samelib/samed1.lib/B_1.a");
Compile (Source => "/usr/same/example/samelib/samed1.lib/P_2.a");
Compile (Source => "/usr/same/example/samelib/samed1.lib/B_2.a");
Compile (Source => "/usr/same/example/samelib/samed1.lib/P_3.a");
Compile (Source => "/usr/same/example/samelib/samed1.lib/B_3.a");
```

Diagnostics

An error is reported and no action is taken if *module_name* is not an abstract or definitional module or does not exist in the library, or if the library is not valid or is locked.

8.7 sde.purge

Command

sde.purge - remove out of date/unused files from an SDE library

Syntax

sde.purge [*pathname*]

Description

sde.purge will empty an existing SDE library of all obsolete or unused files. The command will remove all out of date (due to recompilation for example) or unused files (compiler temporary files or files associated with modules that have been removed via **sde.rm**) along with the library state backup file **samedl.dat.back** in the **samedl.lib** directory from the library associated with *pathname*; if *pathname* is not specified, then the SDE library will default to the curet working directory.

Note that, because **sde.purge** removes the library state backup file **samedl.dat.back**, an SDE library may not be restored back to its prior state once a purge is performed. Normally, library restoration would be accomplished by renaming the **samedl.dat.back** file to **samedl.dat** in the **samedl.lib** directory for the library. For example:

```
%cd pathname/samedl.lib
%ls samedl.dat*
samedl.dat      samedl.dat.back
%rm samedl.dat
%mv samedl.dat.back samedl.dat
```

Examples

The following sequence of commands purges the library contained in the directory **/home/samedl**.

```
%cd /home/samedl
%sde.purge
```

The following command does the same thing:

```
%sde.purge /home/samedl
```

Diagnostics

An error is reported and no action is taken if *pathname* does not specify a valid, unlocked SDE library.

8.8 sde.rm

Command

sde.rm - remove a SAMeDL module from a library

Syntax

sde.rm [*options*] *module_name* ...

Options

- | | |
|---------------------------------|--|
| -force | Suppress the confirmation prompt and force deletion |
| -library <i>pathname</i> | Operate in the SDE library <i>pathname</i> . If not specified, will default to current working directory |

Description

sde.rm removes all information and related interface files associated with the named module(s).

Unless the **-force** option is specified, the user will be issued a confirmation prompt for each module to be removed. The user may respond with a **y** (or **Y**) if the module should be deleted; any other response will result in the module being retained.

Examples

The following sequence of commands removes the unit **abstract_mod** from the SDE library present in the directory **/home/samedl**.

```
%cd /home/samedl
%sde.rm abstract_mod

sde.rm:   Delete ABSMODULE abstract_mod?   [N]: y
```

The following command does the same thing but eliminates the confirmation prompt:

```
%sde.rm -l /home/samedl -f abstract_mod
```

Diagnostics

An error is reported and no action is taken if *module_name* does not exist in the library, or if the library is not valid or is locked.

8.9 sde.rmlib

Command

sde.rmlib - remove an SDE library

Syntax

sde.rmlib [*pathname*]

Description

sde.rmlib removes all information in the SDE library in the directory specified by *pathname* (the current directory is the default). It deletes all the files in the SDE library directory **samedl.lib**, and then removes the directory.

The user will be issued a confirmation prompt. The user may respond with a y (or Y) if the library should be deleted; any other response will abort the command and retain the library unchanged.

Examples

The following sequence of commands removes the SDE library present in the directory **/home/samedl**.

```
%cd /home/samedl
%sde.rmlib

sde.rmlib: Delete ./samedl.lib? [N]: y
```

The following command does the same thing:

```
%sde.rmlib /home/samedl

sde.rmlib: Delete /home/samedl/samedl.lib? [N]: y
```

Diagnostics

An error is reported and no action is taken if the library is not valid or is locked.

Index

A

Ada application program 16-22, 24-25
Ada package file 5, 6, 16, 23, 24

C

C file 11
C/ESQL file 5, 6, 11, 16, 23, 24
Com file 11
Common errors 30
Compiler directive 26-28
Connect server statement 29-30
Context clause 5
Creating a database 13
Creating an SDE library 6, 8, 13-14, 24, 38

D

Database connection 15, 29
Document references 2-3

I

Implementation dependent features 29-30
Interface files 5, 6, 10, 16, 23, 24

L

Language limitations 29
Library locking 10-11

M

Module 5, 10
Module manager 6

N

names.dbc 6, 9

O

Object code file 5, 6, 11, 16, 23, 24
ORACLE_HOME 14, 24
Owner directive 27-28

R

Reference directive 26-27
Restoring an SDE library 9, 31, 43

S

SAMeDL compiler 5, 14-16, 23-28, 34-35
SAMeDL extensions 29-30
SAMeDL standard packages 11, 16

samedl.dat 6, 8
samedl.dat.back 6, 9
samedl.lib 7, 10, 11, 24
samedl.lock 10-11
samedl.tmp 11
SDE library 5, 6, 7-11, 13, 23
sde.cleanlib 6, 8, 36
sde.creatar 6, 17, 37
sde.creatlib 6, 8, 13, 24, 38
sde.ls 6, 16, 39-40
sde.mkscript 6, 17, 41-42
sde.purge 6, 31, 43
sde.rm 6, 44
sde.rmlib 45
Separate compilation 5, 23, 26-27
Syntax conventions 2

T

Tool limitations 31